PRÁCTICA 1 – GEOMETRÍA COMPUTACIONAL – 2023

PABLO FERNÁNDEZ DEL AMO

1. INTRODUCCIÓN:

El objetivo de estos ejercicios es aplicar los conceptos de teoría de la información, como el código Huffman y el Primer Teorema de Shannon, para analizar y comparar la eficiencia de codificación de dos alfabetos diferentes, el inglés y el español, a través de muestras de texto representativas de cada idioma. Además, se busca practicar la codificación y decodificación de palabras utilizando los códigos obtenidos y evaluar su eficiencia en términos de longitud de bits requeridos para la transmisión.

2. MATERIAL USADO:

En estos ejercicios se utilizó una muestra de texto en inglés, “GCOM2023 pract1 auxiliar eng.txt”, y una muestra de texto en español, “GCOM2023 pract1 auxiliar esp.txt”, para aplicar los conceptos de teoría de la información y analizar la eficiencia de codificación de los alfabetos correspondientes. Se usaron los alfabetos S_{Eng} = {a, b, c, ..., Z, !, ?, −, /n, , 0, ..,9} y S_{Esp}= = {a,á, b, c, ...,ñ, ..., Z,¡, !,¿, ?, −, /n, , 0, ..,9} que incluyen letras, signos de puntuación y números, y se aplicó el algoritmo de Huffman para obtener los códigos binarios correspondientes a cada símbolo, la longitud media de cada alfabeto codificado y la entropía de estos. Con estos datos, se ha aplicado y comprobado la validez para este caso concreto del Primer Teorema de Shannon. Además, se usó este algoritmo para codificar una palabra para ambos idiomas, y se comparó la eficiencia de longitud frente al código binario usual. También se decodificó una palabra en inglés utilizando su código binario correspondiente. Todo esto fue realizado a través del lenguaje de programación Python, partiendo de la plantilla dada “GCOM2023-practica1 plantilla.py” y definiendo las siguientes funciones: tabla_codigos, que a partir de un árbol de huffman dado extrae en forma de diccionario el código asociado a cada símbolo del alfabeto en cuestión; codificar_es y codificar_en, que recibe como entrada un texto alfa numérico y devuelve su codificación a través del algoritmo de Huffman, haciendo uso de la plantilla y la función tabla_codigos; y la función decodificar_es y decodificar_en, que realiza la labor inversa a la de codificar.

3. RESULTADOS:

i) A partir de las muestras dadas, hallar el código Huffman binario de S_{Eng} y S_{Esp}, y sus longitudes medias L(S_{Eng}) y L(S_{Esp}). Comprobar que se satisface el Primer Teorema de Shannon:

Se ha tenido en cuenta para realizar la codificación de los alfabetos, que lo caracteres pertenecientes pero que no aparecen en el texto correspondiente tienen frecuencia 0. Con ello, se obtienen las siguientes codificaciones.

Para S_{Eng}:

{'9': '10011111110101111110', 'E': '100111111101011111111', 'F': 1001111111011111110', '1': '10011111110111111111', 'Z': '100111111100111110', 'O': '100111111100111111', 'W': '10011111110100111110', '2': '100111111101001111','7': '10011111110101111110', '8': '10011111110111111110', '6': '10011111110011110', 'R': '100111111101001110', '5': '100111111101011110', 'Q': '100111111101111110', 'X': '10011111111001110', 'P': '10011111110100110', '3': '100111111101011110', 'L': '100111111101111110', '0': '100111111100110', 'N': '10011111111010010', '4': '10011111111010110', 'K': '10011111111011110', '-': '100111111110010', 'J': '100111111101000', '?': '100111111101010', 'G': '100111111101110', '!': '10011111111000', 'Y': '100111111101100', 'U': '100111111101101', '“': '100111111111', 'q': '10011110000', 'H': '10011110001', ')': '10011111100', 'C': '10011111101', '”': '10011111110', 'V': '11101001110', 'j': '11101001111', '\n': '10011111010', '(': '10011111011', 'M': '100111001', 'I': '100111100', 'x': '100111010', 'z': '100111011', 'D': '111010010', 'A': '01111100', 'B': '01111101', 'S': '00100110', 'b': '00100111', 'T': '11101000', 'v': '0010000', 'k': '0010001', ',': '0010010', '.': '0111111', 'y': '1110101', 'w': '011110', 'g': '100110', 'f': '111011', 'p': '00101', 'u': '00110', 'd': '00111', 'm': '01110', 'c': '10010', 'l': '11100', 'r': '11110', 'h': '11111', 's': '0100', 'i': '0101', 'o': '0110', 'n': '1000', 't': '1010', 'a': '1011', 'e': '000', ' ': '110'}

Para S_{Esp}:

{'9': '101101101001111010', 'P': '101101101001111011', 'Ó': '101101101001110111010', '5': '10110110100111011101 1', '4': '10110110100010', 'Ñ': '10110110100011', 'N': '1011011010011010', '2': '1011011010011011', '1': '101101101001110111000', 'J': '10110110100111011100 1', 'Í': '1011011010000111000', 'U': '1011011010000111001', '-': '10110110100110011100 0', 'F': '10110110100110011100 1','H': '1011011010010011100 0', '¡': '10110110100100111001', '¿': '10110110100111011000','A': '1011011010011101100 1', 'K': '10110110100101100', 'X': '10110110100101101', '0': '10110110100111100', 'ú': '101101101000011101', '?': '10110110100110011101', 'G': '1011011010010011101', '!': '1011011010011101101', 'Ú': '1011011010010111', 'R': '1011011010011111', 'ü': '1011011010011101111', 'Ü': '10110110100001111', 'w': '1011011010011001111', 'W': '101101101001001111', 'k': '1011011010000110', '7': '101101101001100110', 'É': '10110110100100110', 'Z': '10110110100111010', 'Á': '101101101000010', 'Y': '10110110100110010', 'O': '1011011010010010', '3': '1011011010011100', 'ñ': '10110110100000', '8': '1011011010011000', 'Q': '101101101001000', '6': '101101101001010', 'V': '10110110101', ')': '1011001100', 'E': '1011001101', '(': '1011001110', 'é': '1011001111', 'x': '1011011011', '\n': '1011011000', 'C': '1011011001', '"': '011111100', 'I': '011111101','L': '101100100', 'D': '101100101', 'M': '111011000', 'z': '111011001', 'v': '01101100', 'í': '01101101', 'S': '111011100', 'B': '111011101', 'T': '01111111', 'q': '10110111', 'y': '11101101', 'ó': '0110100', 'j': '0110101', ',': '11101111', 'á': '0110111', 'f': '0111100', 'b': '0111101', '.': '0111110', 'h': '1011000', 'g': '1011010', 'p': '111010', 'm': '01100', 't': '01110', 'u': '10111', 'c': '11100', 'd': '11110', 'r': '11111', 'l': '0010', 'i': '0011', 'o': '1000', 'n': '1001', 's': '1010', 'e': '000', 'a': '010', ' ': '110'}

El Primer Teorema de Shannon enuncia que el algoritmo de Huffman acota la longitud media del código simbólico según: $H(C) \leq L(C) < H(C) + 1$.

Se tiene que la longitud media y la entropía de S_{Eng} son $L(S_{Eng})= 4.438175270108043$ y $H(S_{Eng})= 4.412184771080396$, mientras que las de S_{Esp} son $L(S_{Esp})= 4.372398685651698$ y $H(S_{Esp})= 4.338300577316107$.

Por lo tanto, ambos alfabetos satisfacen el primer teorema de Shannon.

ii) Utilizando los códigos obtenidos en el apartado anterior, codificar la palabra cognada X = "dimension" para ambas lenguas. Comprobar la eficiencia de longitud frente al código binario usual.

La palabra "dimensión" codificada para el inglés es:
00111010101110000100001000101011010 00
La palabra "dimensión" codificada para el español es:
11110001101100000100110100011100010 01
La palabra "dimensión" codificada en binario es:
1100100 1101001 1101101 1100101 1101110 1110011 1101001 1101111 1101110

Por tanto, la longitud de la cadena utilizando el código binario es 1.9189189189189189 mayor que utilizando Huffman

iii) Decodificar la siguiente palabra del inglés: "0101010001100111001101111000101111111010101000111 0".

Decodificando 0101010001100111001101111000101111111010101000111 0 del inglés se obtiene la palabra: isomorphism

4. CONCLUSIÓN:

En conclusión, la técnica de codificación Huffman binaria resulta efectiva para la compresión de información en idiomas naturales, como el inglés y el español, lo que permite mejorar la eficiencia de transmisión de datos. A partir de la aplicación de esta técnica, se obtuvieron los códigos Huffman binarios para cada idioma y se calculó su longitud media, lo cual demostró que se cumple el Primer Teorema de Shannon. Además, al codificar una palabra cognada utilizando los códigos Huffman binarios, se logró una longitud de codificación menor en comparación con el código binario estándar en ambos idiomas, lo que indica una mayor eficiencia en la transmisión de información. Por último, se comprobó que la decodificación utilizando el código Huffman binario es efectiva, lo que destaca su utilidad en la transmisión y almacenamiento de datos en diferentes áreas.

5. **ANEXO:**
   Código de Python utilizado:

```
"""
Práctica 1. Código de Huffmann y Teorema de Shannon
"""

import os
import numpy as np
import pandas as pd
import math
#### Vamos al directorio de trabajo
os.getcwd()
#os.chdir('D:\Documents\MATEMÁTICAS UCM\4º MATEMÁTICAS 22-23\2 CUATRI\GEOMETRÍA
COMPUTACIONAL\Practicas\Practica 1')
#files = os.listdir('D:\Documents\MATEMÁTICAS UCM\4º MATEMÁTICAS 22-23\2 CUATRI\GEOMETRÍA
COMPUTACIONAL\Practicas\Practica 1')

with open('GCOM2023_pract1_auxiliar_eng.txt', 'r',encoding="utf8") as file: #r sirve para
leer, si quiero escribir es w. el encoding es siempre el mismo
      en = file.read()

with open('GCOM2023_pract1_auxiliar_esp.txt', 'r',encoding="utf8") as file:
      es = file.read()


#### Contamos cuantos caracteres hay en cada texto
from collections import Counter
tab_en = Counter(en)
tab_es = Counter(es)
#Aquí modificamos la plantilla para tener también una codificación de los caracteres que
pertenecen al lenguaje pero no aparecen en el texto
SEng = ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'm', 'n', 'o', 'p',
'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', 'A', 'B', 'C', 'D', 'E', 'F', 'G', 'H',
'I', 'J', 'K', 'L', 'M', 'N', 'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
'!', '?', '-', '\n', ',', '”','“', '(',  ')', '.', '0', '1', '2', '3', '4', '5', '6', '7',
'8', '9']
SEsp = ['a', 'á', 'b', 'c', 'd', 'e', 'é', 'f', 'g', 'h', 'i', 'í', 'j', 'k', 'l', 'm',
'n', 'ñ', 'o', 'ó', 'p', 'q', 'r', 's', 't', 'u', 'ú', 'ü', 'v', 'w', 'x', 'y', 'z', 'A',
'Á', 'B', 'C', 'D', 'E', 'É', 'F', 'G', 'H', 'I', 'Í', 'J', 'K', 'L', 'M', 'N', 'Ñ', 'O',
'Ó', 'P', 'Q', 'R', 'S', 'T', 'U', 'Ú', 'Ü', 'V', 'W', 'X', 'Y', 'Z', '¡', '!', '¿', '?',
'-', '\n', ',','”','“', '(',  ')', '.', '0', '1', '2', '3', '4', '5', '6', '7', '8', '9']
for char in SEng:
    if char not in list(tab_en.keys()):
        tab_en[char] = 0  #los añadimos con frecuencia 0 ya que no aparecen en el texto
for char in SEsp:
    if char not in list(tab_es.keys()):
        tab_es[char] = 0  #los añadimos con frecuencia 0 ya que no aparecen en el texto

#### Transformamos en formato array de los carácteres (states) y su frecuencia
#### Finalmente realizamos un DataFrame con Pandas y ordenamos con 'sort'
tab_en_states = np.array(list(tab_en)) #array con las claves. List de un diccionario
sacauna lista con las claves
```

```
tab_en_weights = np.array(list(tab_en.values())) #array con las frecuencias
tab_en_probab = tab_en_weights/float(np.sum(tab_en_weights))
distr_en = pd.DataFrame({'states': tab_en_states, 'probab': tab_en_probab})
distr_en = distr_en.sort_values(by='probab', ascending=True)
distr_en.index=np.arange(0,len(tab_en_states))

tab_es_states = np.array(list(tab_es))
tab_es_weights = np.array(list(tab_es.values()))
tab_es_probab = tab_es_weights/float(np.sum(tab_es_weights))
distr_es = pd.DataFrame({'states': tab_es_states, 'probab': tab_es_probab })
distr_es = distr_es.sort_values(by='probab', ascending=True)
distr_es.index=np.arange(0,len(tab_es_states))

##### Para obtener una rama, fusionamos los dos states con menor frecuencia
distr = distr_en
''.join(distr['states'][[0,1]])

### Es decir:
states = np.array(distr['states'])
probab = np.array(distr['probab'])
state_new = np.array([''.join(states[[0,1]])])   #Ojo con: state_new.ndim
probab_new = np.array([np.sum(probab[[0,1]])])   #Ojo con: probab_new.ndim
codigo = np.array([{states[0]: 0, states[1]: 1}])
states =  np.concatenate((states[np.arange(2,len(states))], state_new), axis=0)
probab =  np.concatenate((probab[np.arange(2,len(probab))], probab_new), axis=0)
distr = pd.DataFrame({'states': states, 'probab': probab, })
distr = distr.sort_values(by='probab', ascending=True)
distr.index=np.arange(0,len(states))

#Creamos un diccionario
branch = {'distr':distr, 'codigo':codigo}

## Ahora definimos una función que haga exáctamente lo mismo
def huffman_branch(distr):
    states = np.array(distr['states'])
    probab = np.array(distr['probab'])
    state_new = np.array([''.join(states[[0,1]])])
    probab_new = np.array([np.sum(probab[[0,1]])])
    codigo = np.array([{states[0]: 0, states[1]: 1}])
    states =  np.concatenate((states[np.arange(2,len(states))], state_new), axis=0)
    probab =  np.concatenate((probab[np.arange(2,len(probab))], probab_new), axis=0)
    distr = pd.DataFrame({'states': states, 'probab': probab})
    distr = distr.sort_values(by='probab', ascending=True)
    distr.index=np.arange(0,len(states))
    branch = {'distr':distr, 'codigo':codigo}
    return(branch)

def huffman_tree(distr):
    tree = np.array([])
    while len(distr) > 1:
        branch = huffman_branch(distr)
        distr = branch['distr']
        code = np.array([branch['codigo']])
        tree = np.concatenate((tree, code), axis=None)
```

```
    return(tree)
#obtenemos los árboles de huffman asociados al texto en ingés y español
tree_en = huffman_tree(distr_en)
tree_es= huffman_tree(distr_es)

#Esta función recibe un árbol de huffman y devuelve la tabla de códigos, es deicr, un
diccionario donde las claves con los caracteres y los valores su codificación de Huffman
def tabla_codigos(tree):
    tabla = {}
    for par in tree:
        for item in par.items():
            if len(item[0])==1 :
                tabla[item[0]]=str(item[1])
            else:
                for i in item[0] :
                    tabla[i]= str(item[1])+tabla[i]
    return tabla

#Guardamos las tablas asociadas a cada árbol en las siguientes variables:
tabla_en= tabla_codigos(tree_en)
tabla_es= tabla_codigos(tree_es)

# A continuación, definimos dos funciones que reciben como entrada un string y devuelven
la codificación asociada al inglés o al español de este, codificar_en y codificar_es
respectivamente
def codificar_en(texto):
    codigo=''
    for i in texto:
        codigo+=tabla_en[i]
    return codigo

def codificar_es(texto):
    codigo=''
    for i in texto:
        codigo+=tabla_es[i]
    return codigo

#Para decodificar un código de Huffman, independientemente de que sea el código de un
caracter o un texto, aplicaremo sun procedimiento estudiado en Estructuras de Datos
# Básicamente, como para cada char su prefijo es único ( base de la codificación de
huffman), por cada digito que leemos de código (1 o 0), hacemos un filtro en nuestra tabla
de códigos.
#Es decir, que si leemos un 1 como caracter inicial, descartaremos de nuestra tabla todos
aquellos caracteres cuya codificación empiece por 0.
# Si en vez de estar en el primer dígito estamos en el dígito n, aplicaremos el mismo
razonamiento, eliminando aquellos caracteres cuyo código tiene longitud menor a n.
# Al iterar sobre el código input, nuestro diccionario tendrá en un momento tamaño uno.
Ahí añadmios al resultado la única clave que hay y reseteamos el diccionario, continuando
con el algoritmo hasta llegar al final del código input.
# Primero para el texto en inglés y a continuación para en castellano.

def decodificar_en(codigo):
    aux=tabla_en.copy() #Como los diccionarios son mutables, creamos copia para no
modificar nuestra tabla de códigos
```

```
    n=0  #Vamos almacenando la longitud del código asociado al char que estamos intentando encontrar, cuando encontramos un char cuyo código equivale al leído, reseteamos a 0 y volvemos a realizar el proceso
    resultado=''
    for i in codigo:
        lista_elim=[] #Como no podemos iterar sobre un diccionario al que modificamos su tamaño, me creo una lista con los claves que hay que eliminar
        for j in aux:
            if len(tabla_en[j])<=n:
                lista_elim.append(j)  #Si el código asociado a un caracter es más corto que la longitud de código que llevamos (sin decodificar e iterada), hay que eliminarlo
            else:
                if tabla_en[j][n]!=i:  #Si el dígito de código leído no corresponde al dígito del código asociado a una clave para la misma posición, hay que eliminarlo.
                 lista_elim.append(j)
        for k in lista_elim:
            del aux[k]
        n+=1 #Aumentamos en 1 la posición del código que estamos leyendo
        if len(aux)==1:   #Cuando solo queda una clave en el diccionario, esta es la buscada, así que la concatenamos a resultado y reseteamos para continuar el proceso
            resultado+=list(aux.keys())[0]
            aux=tabla_en.copy()
            n=0
    return resultado

def decodificar_es(codigo):
    aux=tabla_es.copy()
    n=0
    resultado=''
    for i in codigo:
        lista_elim=[]
        for j in aux:
            if len(tabla_en[j])<=n:
                lista_elim.append(j)
            else:
                if tabla_es[j][n]!=i:
                 lista_elim.append(j)
        for k in lista_elim:
            del aux[k]
        n+=1
        if len(aux)==1:
            resultado+=list(aux.keys())[0]
            aux=tabla_es.copy()
            n=0
    return resultado

# Función que recibe un texto y lo convierte a binario trivial.
def text_a_binario(texto):
    result = ' '.join(format(ord(c), 'b') for c in texto)
    return result


# EJERCICIO 2
```

```
print('EJERCICIO 1:\n A partir de las muestras dadas, hallar el código Huffman binario de SEng y SEsp, y sus longitudes medias L(SEng) y L(SEsp). Comprobar que se satisface el Primer Teorema de Shannon \n')
#Imprimimos ambas tablas de códigos
print(f'La tabla de códigos de SEng es:\n')
for item in tabla_en.items():
    print(f'{item[0]} : {item[1]}')


print(f'La tabla de códigos de SEsp es:\n')
for item in tabla_en.items():
    print(f'{item[0]} : {item[1]}')



long_media_en=0
entropia_en=0
long_media_es=0
entropia_es=0
for indice_fila, valor_fila in distr_en[distr_en['probab']!=0].iterrows():
    long_media_en += len(tabla_en[valor_fila['states']])*valor_fila['probab'] #Calculamos longitudes medias con la fórmula vista en clase
    entropia_en-=valor_fila['probab']*math.log2(valor_fila['probab'])
for indice_fila, valor_fila in distr_es[distr_es['probab']!=0].iterrows():
    long_media_es += len(tabla_es[valor_fila['states']])*valor_fila['probab']
    entropia_es-=valor_fila['probab']*math.log2(valor_fila['probab'])

print(f'\nLa longitud media de SEng, L(SEng), es igual a {long_media_en}; y su entropía, H(SEng); es igual a {entropia_en}')
print(f'La longitud media de SEsp, L(SEsp), es igual a {long_media_es}; y su entropía, H(SEsp), es igual a {entropia_es}')

print('\nEntonces, para ambos alfabetos el Primer Teorema de Shannon se cumple, ya que H(i) <= L(i) < H(i)+1; con i SEng o SEsp')


#EJERCICIO 2:
print('\n\nEJERCICIO 2:\n Utilizando los códigos obtenidos en el apartado anterior, codificar la palabra cognada X =“dimension” para ambas lenguas. Comprobar la eficiencia de longitud frente al código binario usual. \n')

dimension_en=codificar_en('dimension')
dimension_es=codificar_es('dimension')
dimension_bin=text_a_binario('dimension')
print('La palabra dimension codificada para el ingles es:', dimension_en)
print('La palabra dimension codificada para el español es:', dimension_es)
print('La palabra dimension codificada en binario es:', dimension_bin)
print(f'\nPor tanto, la longitud de la cadena utilizando el código binario es {len(dimension_bin)/len(dimension_en)} mayor que utilizando Huffman')


#EJERCICIO 3:
```

```
print('\n\nEJERCICIO 3:\n Utilizando los códigos obtenidos en el apartado anterior,
codificar la palabra cognada X =“dimension” para ambas lenguas. Comprobar la eficiencia de
longitud frente al código binario usual. \n')

print('Decodificando 0101010001100111001101111000101111110101010001110 del inglés se
obtiene la palabra:', decodificar_en('0101010001100111001101111000101111110101010001110'))
```